VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# AI Clash

# AI Clash

2012 Tomáš Worek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Tomáš Worek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: AI Clash
AI Clash

Zásady pro vypracování:

Cílem práce je vytvořit aplikaci simulující bitvy robotů. Program se bude skládat z tvorby robota, tvorby bitevního pole a simulace bitvy.
Návrh robota bude probíhat nejprve nastavením technického vybavení robota v GUI a následně naprogramováním AI pomocí API funkcí specificky vytvořeného jazyka. API funkce umožní reagovat na události jako kolize se zdí, zásah střelou, robot na radaru, apod.
V části tvorby bitevního pole bude možno generovat či vytvářet vlastní bitevní pole.
V poslední části, pak bude možno vybrat bitevní pole a spouštět samotnou simulaci.
Implementace bude probíhat v jazyce C#.

Práce bude obsahovat:
1. Analýzu a popis podobných programů.
2. Specifikaci systému - podporované vlastnosti pro jednotlivé části systému.
3. Specifikaci jazyka pro programování chování.
4. Implementaci systému.
5. Implementaci několika ukázkových robotů a testování systému.

Seznam doporučené odborné literatury:

Ján Hanák: Praktické objektové programování v jazyce C# 4.0, 978-80-87017-07-4, Artax 2009
Ján Hanák: Praktické paralelní programovaní v C# 4.0 a C++, 978-80-87017-06-7, Artax, 2009
Dále dle pokynů vedoucího práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 24. dubna 2012 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2012 .....

Rád bych na tomto místě poděkoval zejména vedoucímu své bakalářské práce Ing. Janu Platošovi Ph.D. za konzultace a rady při vypracování této bakalářské práce.

## Abstrakt

Cílem práce je vytvořit aplikaci simulující bitvy robotů. Program se bude skládat z tvorby robota, tvorby bitevního pole a simulace bitvy. Návrh robota bude probíhat nejprve nastavením technického vybavení robota v GUI a následně naprogramováním AI pomocí API funkcí specificky vytvořeného jazyka. Tento jazyk umožní reagovat na události jako kolize se zdí, zásah střelou, robot na radaru, apod. V části tvorby bitevního pole bude možno generovat či vytvářet vlastní bitevní pole. V poslední části, pak bude možno vybrat bitevní pole a spouštět samotnou simulaci. Implementace bude probíhat v jazyce C#.

**Klíčová slova:** AI Clash, roboti, programovací hra, simulace

## Abstract

Goal of this work is to create application to simulate battles of the robots. Application will contains of creation of a robot, creation of a battlefield and simulation of the battle. Design of the robot will contains robot hardware settings in the GUI and AI programming using API functions of specifically constructed language. In this language will be able respond to events such as collisions with the wall, hit a shot, a robot on the radar, etc. In a section of creating the battlefield will be able to generate or create your own battlefield. In the last section, it will be possible to choose the battlefield itself and run the simulation. Implementation will be in C#.

**Keywords:** AI Clash, robots, programming game, simulation

## Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| AI | – | Artificial Intelligence |
| API | – | Application Programming Interface |
| GUI | – | Graphical User Interface |
| MARS | – | Memory Array Redcode Simulator |
| PLATO | – | Programmed Logic for Automated Teaching Operations |
| RPN | – | Reversion Polish Notation |

# Obsah

# Seznam tabulek

# Seznam obrázků

# Seznam výpisů zdrojového kódu

# 1 Úvod

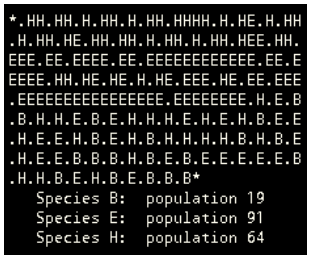
Aplikace AI Clash je simulátor bitev robotů a patří mezi tzv. programovací hry. Uživatelé těchto her nemají možnost řídit či jinak ovlivňovat průběh simulace. Místo toho je v tomto typu aplikací úkolem uživatele v předem jasně daném, nejčastěji specificky vytvořeném, programovacím jazyce naprogramovat umělou inteligenci charakteru (obvykle robota či bakterie), který je pak do dané simulace postaven proti charakterům ostatních uživatelů, ať už za účelem přežití, nasbírání největšího počtu bodů, či splnění jiných cílů.

První část práce je věnována historii programovacích her a představení mezníků. V této části budou představeny hry Darwin, Core War a RobotWar. V druhé části bude popsána specifikace systému, co umožňuje a jaké mají vlastnosti bitevní pole, roboti, powerbally a střely. V následující části pak bude popsán specificky vytvořený jazyk RoboCode, který slouží k naprogramování umělé inteligence robotů. V dalších částech bude postupně popsán překladač, který slouží k překladu z kódu v jazyce RoboCode do interpreterem srozumitelného kódu, následovaný popisem samotného interpreteru. V závěru pak budou shrnuty hlavní body, zhodnoceny dosažené výsledky a proběhne nástin směru, kterým se bude vývoj této práce dále ubírat.

# 2 Historie a hlavní představitelé

## 2.1 Darwin

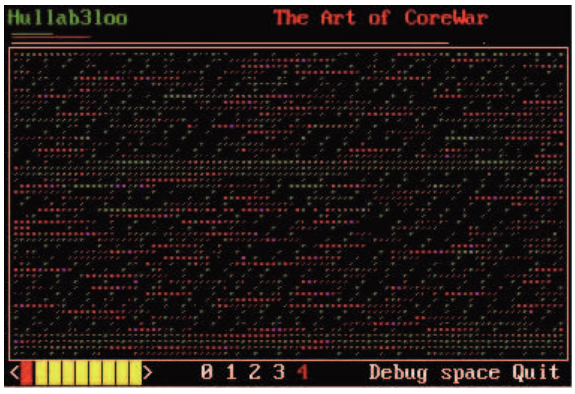
První aplikace, která se řadí do skupiny programovacích her, jménem Darwin, byla vytvořena už v roce 1961 v Bell Labs pány Douglesem McIlroyem, Robertem Morrisem a Victorem Vyssotskym (Metcalf, 2011a). Cílem uživatelů aplikace bylo napsat IBM 7090 program takový, aby byl co nejvíce životaschopným replikátorem v paměti. Pro tento cíl byly k dispozici 3 funkce (test daného místa, zabrání prázdného místa a smazání obsazeného místa) a zpočátku jen 15 instrukcí, jejichž počet se později rozrostl až na 44 instrukcí.




Obrázek 1: Aplikace Darwin

## 2.2 Core War

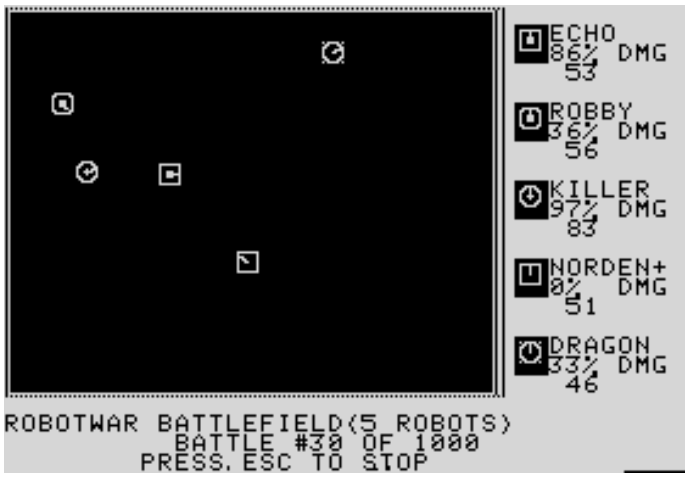
V roce 1984 byla pány D. G. Jonesem a A. K. Dewdneyem představena desetistránková příručka Core War Guidelines (Jones, Dewdney, 1984) popisující programovací jazyk Redcode určený pro virtuální počítač MARS. Ještě téhož roku vydal Kevin A. Bjorke zdrojový kód vytvořený v jazyce C implementující virtuální počítač MARS a o dva roky později, roku 1986, byl v Bostonu uspořádán první mezinárodní turnaj v programovací hře Core War. Turnaje v Core War probíhají v mírně pozměněné formě dodnes.




Obrázek 2: Aplikace Core War

## 2.3 RobotWar

Avšak ještě předtím byla v roce 1981 publikována pod záštitou MUSE Software hra jménem RobotWar, první programovací hra, která simulovala bitvy robotů (Metcalf, 2011b). Byla napsána Silasem Warnerem pro počítačový systém PLATO. První turnaj byl pak uspořádán hned o rok později. Úkolem uživatele bylo naprogramovat umělou inteligenci robota, která by uspěla v souboji proti jiným robotům. Vítězem těchto simulací byl pak poslední přeživší.



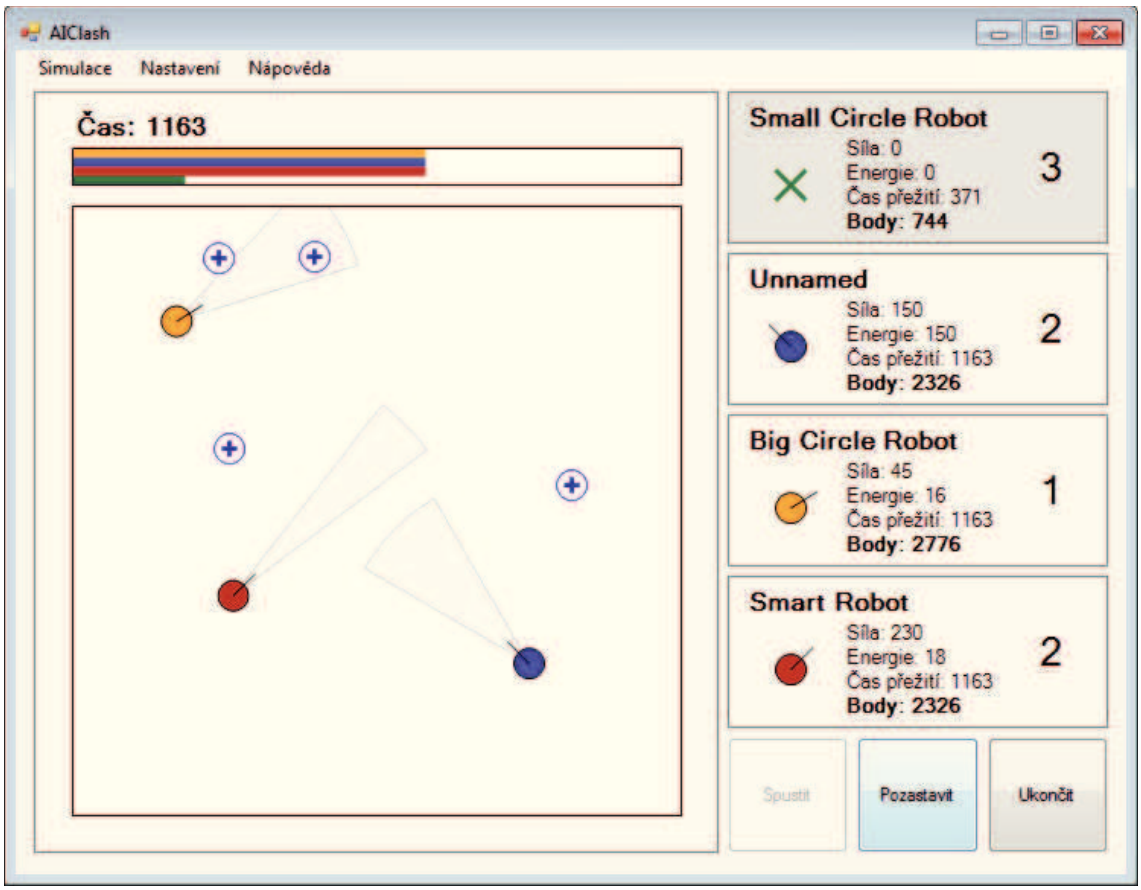

Obrázek 3: Aplikace RobotWar

# 3 Specifikace systému

## 3.1 Simulace

Úkolem simulace v aplikaci AI Clash je vyhodnotit a vizuálně zobrazit průběh bitvy na základě vstupních dat. Vstupními daty jsou v tomto případě jedno bitevní pole, viz kapitola 3.2, a příslušný počet robotů, viz kapitola 3.2.2, účastnících se klání. Uživatel má umožněno spouštět simulaci opakovaně pro různá vstupní data, avšak po spuštění simulace už nemá možnost jakkoli zasáhnout a změnit průběh již započaté simulace. Má však možnost nastavit délku simulace před započetím a v průběhu simulace dokonce měnit rychlost simulace. Délka simulace je udávána v cyklech a možnými hodnotami jsou „500“, „1 000“, „2 000“, „5 000“, či „10 000“. Rychlost simulace je pak možno nastavit na „Rychlá“, „Střední“, nebo „Pomalá“, čemuž odpovídají rychlosti 100 cyklů za vteřinu, 10 cyklů za vteřinu a 1 cyklus za vteřinu.

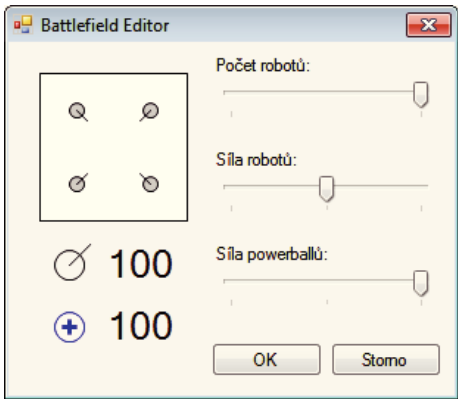
Snímek z průběhu bitvy je zobrazen na Obrázku 4.




Obrázek 4: Aplikace AI Clash

## 3.2 Bitevní pole

Bitevní pole je místo, na kterém se odehrává simulace jednotlivých klání. Při tvorbě bitevního pole má uživatel možnost zvolit počet robotů, účastnících se klání, počáteční sílu robotů a také sílu powerballů.




Obrázek 5: Battlefield Editor

### 3.2.1 Rozměr, osy a úhly

Bitevní pole má z nadhledu čtvercový tvar o rozměrech 400 x 400 pixelů a je po okrajích bezprostředně ohraničeno neprůchodnými a neprůstřelnými zdmi. Osa x bitevního pole je lineární, má počátek v levém horním rohu a směrem vpravo nabývá kladných hodnot. Osa y je také lineární, také má počátek v levém horním rohu, avšak kladných hodnot nabývá ve směru dolů. Úhly v této soustavě souřadnic jsou vyjádřeny ve stupních a nabývají hodnot z intervalu (-180 ; 180⟩. Při práci s hodnotami mimo tento interval je hodnota automaticky upravována přičítáním či odečítáním periody 360°. Úhel 0° je orientován ve směru nahoru, resp. v záporném směru osy y. Úhel 90° je orientován vpravo, resp. v kladném směru osy x. Úhel tedy roste po směru hodinových ručiček.

### 3.2.2 Roboti

Roboti jsou základními bitevními prvky v poli a při simulaci jsou od sebe navzájem odlišeni barvou. Roboti účastnící se klání jsou omezeni pohybem pouze v oblasti bitevního pole ohraničeného zdmi. Počet robotů, účastnících se klání, je volen při tvorbě bitevního pole, viz Obrázek 5. Uživatel má na výběr z počtu „2“ a „4“ robotů a má také možnost pro dané klání nastavit sílu robota z možností „50“, „100“ a „150“, viz Obrázek 5. Další popis robota a jeho vlastností jsou uvedeny v kapitole 3.3 a odpovídajících podkapitolách.

### 3.2.3 Powerbally

Powerbally jsou z nadhledu kruhové objekty o poloměru 10 pixelů, které se objevují v bitevním poli na náhodných pozicích v průběhu simulace a jež dodávají sílu robotům. Tyto objekty mají po vytvoření pevně danou pozici a dále se již nepohybují. V nastavení bitevního pole má uživatel možnost nastavit sílu powerballů, viz Obrázek 5. Na výběr jsou možnosti „0“, „50“ a „100“ jednotek síly. Při volbě „0“ bude simulace probíhat bez powerballů. Pokud uživatel vybere jednu z dalších možností, bude při sebrání powerballu tomuto robotu přičtena síla dle nastavení.

### 3.2.4 Střely

Střely jsou kulové objekty o poloměru 2 pixely a rychlostí 10 pixelů za cyklus, kterými robot škodí ostatním robotů. První případ ukončení existence střely je při střetu střely se zdí. V takovém případě střela jednoduše přestane existovat. Druhým případem je střet 2 střel, ve kterém obě střely přestanou existovat a to bez ohledu na sílu jednotlivých střel. Třetím a posledním případem je střet střely s robotem. Tehdy se odečte od síly robota síla střely. Ve všech případech nemá zrušení střely žádný jiný následek na okolní objekty, které se střelou nepřišly do styku, kromě případného přičtení bodů robotovi vlastnícímu střelu.

## 3.3 Robot

Robot je základní bojovou jednotkou. Uživatel má z hardwarové části robota možnost nastavit rozměr(šířka a dosah) radaru, softwarová část robota pak dává uživateli povinnost naprogramovat chování robota v jazyce RoboCode, který je popsaný v kapitole 5. Po úspěšném překladu (kompilaci) je robot připraven ke klání.

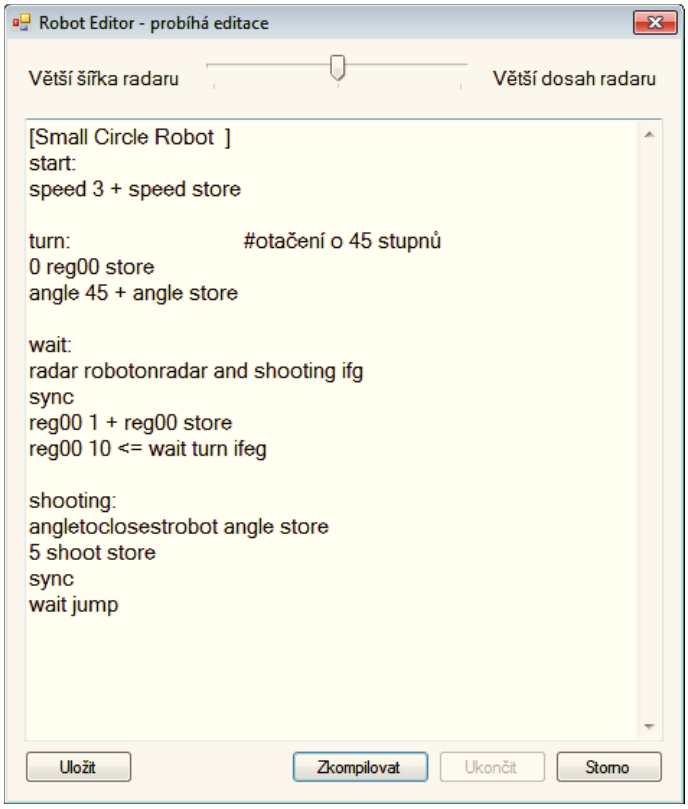
### 3.3.1 Stavy

Robot se může vyskytovat ve třech stavech:

- Živý - V tomto stavu je robot schopen vykonávat libovolné akce a je v každém cyklu simulace oceněn 2 body.
- Zmrzlý - V tomto stavu je robot neschopen akce, avšak i přesto se stále účastní klání. Je však v každém cyklu oceněn pouze 1 bodem.
- Mrtvý - V tomto stavu se robot již dále neúčastní klání a není mu tedy připočítáváno bodové ohodnocení.

### 3.3.2 Rozměr a rychlost

Robot má z nadhledu kruhový tvar o poloměru 10 pixelů. Rychlost robota je udávána v celočíselných hodnotách vyjadřujících počet pixelů za cyklus. Tato rychlost se pohybuje v intervalu $\langle$ -5 ; 5 $\rangle$.




Obrázek 6: Robot Editor

### 3.3.3 Síla a Energie

Síla robota udává jeho životaschopnost v bitvě. Pokud robotu klesne hodnota na 0, pak je robot mrtev a dále se boje neúčastní.

Energie je hodnota, kterou může robot spotřebovat a využít ji k pohybu či akci. Tato energie se navíc po každém cyklu navýší o 1 jednotku. Úbytek energie až na nulovou hodnotu nemá žádný omezující vliv na robota, avšak není možno při nedostatku energie využívat příkazů, jež energii spotřebovávají (změna směru pohybu, střelba, apod.).

### 3.3.4 Radar

Radar slouží k prohledávání okolí. Uživatel má na výběr 3 možnosti nastavení radaru, viz obr.5. Nastavením možnosti „Větší šířka radaru“ bude šířka radaru robota nastavena na 75° a dosah na 80 pixelů. Možnost „Větší dosah radaru“ nastaví šířku radaru na 15° a dosah na 160 pixelů. Výběrem střední možnosti je pak nastavena šířka radaru na 30° a dosah na 125 pixelů.

### 3.3.5 Firmware

Firmware je softwarová část robota, obsahující překladačem přeloženou uživatelem vytvořenou umělou inteligenci robota. Uživatel nejprve naprogramuje chování robota v jazyce RoboCode. Zdrojový kód v tomto jazyce je následně překladačem přeložen do vnitřního kódu (firmware) robota. Tento kód je pak v průběhu simulace interpretován. Více k překladači a interpreteru v kapitole 6, resp. v kapitole 7.

### 3.3.6 Registry

Registry jsou místa v paměti robota a slouží mu k záznamu a vyčítání dat.

# 4 Cyklus simulace

Cyklus simulace je jednotka průběhu simulace. Celá simulace probíhá v několika stovkách cyklů (dle nastavení). Každý cyklus se sestává z následujících částí:

1. Vygenerování powerballu, viz kapitola 4.1.
2. Interpretace robotů, viz kapitola 4.2.
3. Aktualizace polohy objektů, viz kapitola 4.3.
4. Vyřešení kolizí, viz kapitola 4.4.
5. Aktualizace robotů v kolizi, viz kapitola 4.4.
6. První smazání zrušených objektů, viz kapitola 4.5.
7. Vytvoření nových objektů, viz kapitola 4.5.
8. Druhé smazání zrušených objektů, viz kapitola 4.5.
9. Aktualizace síly a energie robotů, viz kapitola 4.6.
10. Nastavení času přežití a pořadí robotům, viz kapitola 4.6.
11. Ověření ukončení simulace, viz kapitola 4.7.

## 4.1 Vygenerování powerballu

Powerball je vygenerován na náhodném místě v bitevním poli, pokud jsou splněny následující podmínky:

- Číslo cyklu je rovno 0 nebo násobku desetiny délky simulace.
- Síla powerballu při tvorbě bitevního pole byla zvolena 50, nebo 100.
- Nebylo dosaženo maximálního počtu powerballů v poli. Tento počet je pevně nastaven na 5.

## 4.2 Interpretace robotů

Interpretace jednotlivých robotů probíhá postupně, avšak akce ovlivňující okolí jsou aktualizovány až v dalších částech cyklu, aby nebyli dříve interpretovaní roboti zvýhodněni oproti následně interpretovaným. Interpretace každého robota probíhá v těchto krocích:

1. Přičtení bodů robotovi dle stavu, ve kterém se nachází. Robot ve stavu „Živý“ je oceněn 2 body, robot ve stavu „Zmrzlý“ jedním bodem a robot ve stavu „Mrtvý“ není oceněn žádným bodem.

2. Pokud je robot „Zmrzlý“ nebo „Mrtvý“, ukončí se interpretace tohoto robota a přejde se k interpretaci dalšího.
3. Zjištění, kolik má robot energie. Podle dostupné energie bude robotovi při interpretaci kódu povoleno či zakázáno provádět akce spotřebovávající energii.
4. V tomto kroku se provádí dle rychlosti procesoru robota počet interpretovacích kroků firmwaru robota. Rychlost procesoru každého robota je pevně nastavena na 20 kroků.

Při interpretaci robotů jsou zachytávány definované i nedefinované výjimky vyvolané interpreterem firmwaru. Po zachycení libovolné výjimky je stav robota změněn na „Zmrzlý“. Interpreter firmwaru a jím definované výjimky jsou popsány v kapitole 7.

## 4.3 Aktualizace polohy objektů

Aktualizace polohy objektu se týká pohybujících se objektů, tedy robotů a střel. Momentální pozice je zálohována a poté změněna dle směru a rychlosti objektu.

## 4.4 Vyřešení kolizí

Jelikož je při vytváření powerballu v prvním kroku cyklu počítáno s rozměrem bitevního pole a je řešena kolize se stávajícími powerbally a jelikož se powerbally nepohybují, nemůže tedy dojít dodatečně k jejich vzájemné kolizi či kolizi se zdí, a tak tyto kolize nemusejí být řešeny. Avšak zbývající jednotky v bitevním poli jsou schopné pohybu, musí se tedy řešit všechny ostatní kolize, které jsou popsány v následujících podkapitolách. Odečtení či přičtení jednotek síly a energie je fakticky provedeno až v části cyklu, která má na starosti aktualizaci síly a energie robotů, nikoli ihned při řešení kolizí. Stejně tak, pokud má být objekt při řešení kolizí odstraněn, je pouze přiřazen do seznamu objektů určených ke smazání a smazán v pozdějších krocích cyklu.

### 4.4.1 Kolize střely se zdí

Při této kolizi je střela přiřazena do seznamu objektů určených ke smazání. Zeď není nijak poškozena.

### 4.4.2 Kolize robota se zdí

Při této kolizi je robotu odečteno 5 jednotek síly a je navrácen na minulou pozici.

### 4.4.3 Kolize robota s robotem

Při této kolizi jsou oba roboti přidáni do seznamu kolidujících robotů. Tento seznam se prochází a řeší až po vyřešení všech ostatních kolizí a sesbírání všech kolidujících robotů, aby se předešlo zvýhodnění některého z robotů.

### 4.4.4 Kolize robota se střelou

Tato kolize je brána jako zásah robota střelou. Robotu je odečtena síla rovnající se síle střely. Vlastníkovi střely je dále přičten ke stávajícím bodům desetinásobek síly střely. Nakonec je střela přiřazena do seznamu objektů určených ke smazání.

### 4.4.5 Kolize robota s powerballem

Při této kolizi je robotu přidána energie rovnající se síle powerballu. Powerball je poté přidán do seznamu objektů určených ke smazání.

### 4.4.6 Kolize střely se střelou

Při této kolizi jsou obě střely přiřazeny do seznamu objektů určených ke smazání. Není tedy brán ohled na sílu jednotlivých střel.

### 4.4.7 Kolize střely s powerballem

Tento střet není brán jako kolize, jelikož pomyslná výška powerballu je níže než pásmo putování střel. Střela tedy daný powerball přeletí a ke kolizi nedojde, ač to tak z nadhledu vypadá.

### 4.4.8 Vyřešení kolidujících robotů

Po vyřešení všech kolizí a nalezení všech kolidujících robotů se přistoupí k vyřešení kolidujících robotů. Všem takovýmto robotům je odečteno 5 jednotek síly a jsou navrácení na předchozí pozici.

## 4.5 Smazání a vytvoření objektů

Tato fáze se skládá ze tří po sobě jdoucích částí:

1. První smazání zrušených objektů.
2. Vytvoření nových objektů.
3. Druhé smazání zrušených objektů.

### 4.5.1 První smazání zrušených objektů

V této části jsou smazány všechny objekty, jež byly v rámci řešení kolizí přidány do seznamu objektů určených ke smazání. Tento seznam je po smazání objektů vyprázdněn.

### 4.5.2 Vytvoření nových objektů

V této části jsou vytvořeny nové objekty v bitevním poli. Jelikož v průběhu klání není možno vytvářet roboty a o tvorbu powerballu se stará jiná část cyklu, jedinými takto vytvořenými objekty mohou být střely. Ty se přiřazují do tohoto seznamu při interpretaci firmwaru robotů a vytvářeny jsou až v této části, aby se předešlo zvýhodnění dříve interpretovaných robotu před později interpretovanými. Jelikož je možné, že při vytváření těchto střel došlo k dalším kolizím, jsou analogicky ke kapitole 4.4, zjištěny a řešeny kolize, avšak pouze kolize s těmito novými střelami. Seznam s nově vytvořenými objekty je po provedení vyprázdněn.

### 4.5.3 Druhé smazání zrušených objektů

V případě, že při vytváření nových objektů došlo ke kolizi či kolizím, které si vyžadují smazání některých objektů, resp. se v seznamu objektů určených ke smazání vyskytují další objekty, jsou tyto objekty smazány a seznam vyprázdněn.

## 4.6 Aktualizace parametrů robotů

Tato fáze probíhá ve dvou krocích:

1. Aktualizace síly a energie robotů.
2. Nastavení času přežití a pořadí robotům.

### 4.6.1 Aktualizace síly a energie robotů

V této části se každému robotovi účastnícímu se klání aktualizují hodnoty síly a energie. Pokud síla klesla na hodnotu menší nebo rovnu nule je robot označen jako „Mrtvý“. Síla robota může nabývat hodnot od 0 do 250. Energie pak od 0 až po aktuální hodnotu síly robota.

### 4.6.2 Nastavení času přežití a pořadí robotům

Robotům, kteří jsou ve stavu „Živý“ nebo „Zmrzlý“, je aktualizován čas přežití na index probíhaného cyklu.

Pořadí robotů je počítáno speciálním algoritmem, viz Výpis 1, který je sestaven na základě „Selection sortu“ beroucí v úvahu stav robota a počet bodů. Tento algoritmus je sestaven tak, aby upřednostňoval aktivní roboty, kteří jsou oceňováni, když jejich střela zasáhne jiného robota a/nebo za rychlé ukončení simulace zničením ostatních robotů. Jsou však také vyzdvihovány body i přes úmrtí robota. V jistých případech může tedy robot, který byl v boji dostatečně aktivní, ale před ukončením simulace zemřel, vyhrát.

```
List<Robot> list = robots;
short n = countOfRobots;
short i;
short max;
```

```
short prize = 1;

for (i = 0; i < n; i++)
{
    max = i;
    for (short j = (short)(i + 1); j < n; j++)
    {
        if ( list [j ]. points > list [max].points ||
             list [j ]. points == list [max].points && list [j ]. status < list [max].status))
        {
            max = j;
        }
    }
    if (max != i)
    {
        Robot tmp = list [i ];
        list [i ] = list [max];
        list [max] = tmp;
    }
    // udělení ocenění
    if (i == 0)
    {
        list [i ]. prize = prize;
    }
    else
    {
        list [i ]. prize = ( list [i − 1].points == list [i ]. points && list [i − 1].status == list [i ].
            status) ? prize : ++prize;
    }
}
```

Výpis 1: Algoritmus oceňující roboty.

## 4.7 Ověření ukončení simulace

Ukončení simulace proběhne při splnění jakékoli z těchto podmínek:

- Index cyklu je roven délce simulace.
- Bitevní pole neobsahuje ani jednoho robota ve stavu „Živý“ a žádné střely.
- Bitevní pole obsahuje jediného robota ve stavu „Živý“, žádného robota ve stavu „Zmrzlý“ a žádné střely.

V případě, že je simulace ukončena před dosažením úplné délky simulace, tedy v případě splnění 2. či 3. podmínky, jsou robotům, které nejsou ve stavu „Mrtvý“ připočítány body za neproběhnuté cykly. Robotu ve stavu „Živý“ je k dosavadním bodům připočten dvojnásobek počtu zbývajících cyklů, robotům, ve stavu „Zmrzlý“ je k dosavadním bodům připočten počet zbývajících cyklů.

# 5 Jazyk RoboCode

Jazyk RoboCode je speciálně navržený jazyk vycházející z jazyka RoboTalk, avšak není s ním zpětně kompatibilní, jelikož je přizpůsoben pro systém v aplikaci AI Clash. RoboCode je zásobníkovým programovacím jazykem dodržujícím postfixovou notaci, nebo též RPN. Jazyk je navržen tak, aby byl co nejsnadnější k pochopení uživatelem, co nejjednodušší pro interpretaci a přitom dostatečně rychlý.

Tento jazyk slouží k naprogramování umělé inteligence robota a skládá se ze jména robota, komentářů, čísel, labelů, registrů, instrukcí a oddělovačů. RoboCode není case-sensitive, takže je na uživateli, zda bude kód psát pouze malými znaky, pouze velkými znaky, nebo bude používat libovolnou kombinaci malých a velkých znaků.

**Zásobník** Zásobník slouží k dočasnému ukládání dat a pracuje s daty tak, že data uložená jako poslední budou ze zásobníku vyčtena jako první. Ukázka manipulace s daty v zásobníku je popsána Tabulce 1. Tato ukázka pracuje s matematickmi instrukcemi, jenž jsou popsány v Tabulce 2.

| **Vstup** | **Zásobník** | | | |
|---|---|---|---|---|
| 5 | 5 | | | |
| 20 | 5 | 20 | | |
| + | 25 | | | |
| -1 | 25 | -1 | | |
| 7 | 25 | -1 | 7 | |
| 45 | 25 | -1 | 7 | 45 |
| sin | 25 | -1 | 5 | |
| * | 25 | -5 | | |
| abs | 25 | 5 | | |
| / | 5 | | | |

Tabulka 1: Zásobník

## 5.1 Jméno robota

Jméno robota je volitelné. Pokud je vyplňováno, musí být umístěno na prvním řádku kódu v hranatých závorkách bez dalšího kódu v řádku.

## 5.2 Komentáře

Komentáře slouží uživateli k záznamu nejrůznějších poznámek přímo do kódu. Začínají znakem „#“ a jsou ukončeny koncem řádku.

## 5.3 Čísla

Čísla v RoboCodu jsou celočíselná čísla z intervalu ⟨ -29 999 ; 29 999 ⟩. Je povoleno bezprostředně před číslem použít znaménka kladu a záporu, „+“ resp. „-“.

## 5.4 Label

Labely jsou textové štítky sloužící jako reference k místu kódu o maximální délce 30 znaků. Skládají se ze 2 typů: reference a odkaz. Obojí se skládá ze stejných znaků, avšak reference je zakončena znakem „:“.

## 5.5 Registry

Registry jsou virtuální místa v paměti robota, jež dovolují uživateli vyčítat či ukládat hodnoty. Registry Angle, Speed a uživatelské registry jsou určeny ke čtení i zápisu. Registry X, Y, Power, Energy, Radar, RobotOnRadar, PowerballOnRadar, AngleToClosest, AngleToClosestRobot, AngleToClosestPowerball, Collision, Hit, AngleToHit jsou určeny pouze ke čtení. Registr Shoot pak slouží pouze pro zápis.

**Registr Angle** určuje úhel směru, ve kterém je robot natočen. Je udáván ve stupních v intervalu ( -180 ; 180 ⟩. Při zápisu do tohoto registru je robotu nastaven směr daný nastavovanou hodnotou. To je ale provedeno pouze v případě, že má robot dostatečnou energii na tento úkon. Potřebná energie se vypočítává z tohoto vzorce:

$$energy = \frac{(speed + 1) * (change)}{9};$$

kde „energy“ je potřebná energie, „speed“ je rychlost robota a „change“ je delta změny směru robota.

**Registr Speed** obsahuje hodnotu velikosti rychlosti pohybu robota. Povoleno je nastavovat rychlost z intervalu ⟨ -5 ; 5 ⟩. Při nastavení hodnoty menší než -5, resp. větší než 5, bude hodnota nastavena na -5, resp. 5. Daná rychlost se však nastaví pouze tehdy, má-li robot dostatek energie. Potřebná energie je rovna velikosti změny rychlosti. Pokud má tedy robot aktuální rychlost 2 pixely za vteřinu a nastavována je rychlost 5 pixelů za vteřinu, je změna rovna 3 a je tedy potřeba 3 jednotek energie.

**Registr Shoot** slouží ke střelbě. Zápis do tohoto registru způsobí střelu o síle zapsané hodnoty, avšak pouze v případě, že má robot energii rovnu zapsané hodnotě. Zasaženému robotu se odečte tato hodnota od výkonu. I když tento registr slouží pouze pro zápis, je možno z tohoto registru také číst, avšak návratová hodnota je vždy „0“.

**Registr X** vrací x-ovou pozici robota vzhledem k bitevnímu poli.

**Registr Y** vrací y-ovou pozici robota vzhledem k bitevnímu poli.

**Registr Power** vrací hodnotu síly robota.

**Registr Energy** vrací hodnotu energie robota.

**Registr Radar** vrací „1“, pokud se na radaru nachází jiný robot či powerball. V opačném případě vrací „0“.

**Registr RobotOnRadar** vrací „1“ pokud se na radaru nachází jiný robot. V opačném případě vrací „0“.

**Registr PowerballOnRadar** vrací „1“ pokud se na radaru nachází powerball. V opačném případě vrací „0“.

**Registr AngleToClosest** vrací úhel k nejbližšímu objektu (robot nebo powerball) na radaru. V případě, že na radaru žádný objekt není, je vrácena „0“.

**Registr AngleToClosestRobot** vrací úhel k nejbližšímu robotovi na radaru. V případě, že na radaru žádný robot není, je vrácena „0“.

**Registr AngleToClosestPowerball** vrací úhel k nejbližšímu powerballu na radaru. V případě, že na radaru žádný powerball není, je vrácena „0“.

**Registr Collisions** vrací „1“ v případě, že je robot v kolizi s jiným robotem. V opačném případě vrací „0“.

**Registr Hit** vrací „1“ v případě, že byl robot zasažen někdy v průběhu posledních 5 cyklů. V opačném případě je vrácena „0“.

**Registr AngleToHit** vrací úhel ke střele v okamžiku zásahu.

**Uživatelské registry** jsou registry, jež slouží uživateli k ukládání hodnot. Těchto registrů je 64 a mají pevně dané názvy skládající se z pěti znaků. První tři znaky jsou „reg“ a zbývající dva znaky označují dvojciferné číslo v hexadecimální soustavě v rozmezí „00“ - „3F“. Příklady názvů takových registrů jsou tedy „reg00“, „reg1B“, či „reg33“.

## 5.6 Instrukce

Instrukce slouží jako příkaz pro interpreter k vykonání určité operace. RoboCode obsahuje celkem 37 instrukcí, matematické jsou popsány v Tabulce 2, logické v Tabulce 3, řídící a ostatní instrukce pak v Tabulce 4.

## 5.7 Oddělovače

Jako oddělovače mezi stránkami slouží znaky: „mezera“ „tabulátor“ „nový řádek“.

| Instrukce | Počet hodnot vstup | výstup | Popis |
|---|---|---|---|
| + | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a součet uloží zpět. |
| - | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a rozdíl **B** a **A** uloží zpět. |
| * | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a součin uloží zpět. |
| / | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a podíl **B** a **A** uloží zpět. |
| % | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a zbytek po celočíselném dělení **B** b **A** uloží zpět. |
| abs | 1 | 1 | Vyčte hodnotu ze zásobníku a její absolutní hodnotu uloží zpět. |
| sqr | 1 | 1 | Vyčte hodnotu ze zásobníku a její druhou mocninu uloží zpět. |
| sqrt | 1 | 1 | Vyčte hodnotu ze zásobníku a její druhou odmocninu uloží zpět. |
| sin | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a součin **B** a sinus(**A**) uloží zpět. |
| cos | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a součin **B** a cosinus(**A**) uloží zpět. |
| tg<br>tan | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a součin **B** a tangens(**A**) uloží zpět. |

Tabulka 2: Matematické instrukce

| Instrukce | Počet hodnot vstup | výstup | Popis |
|---|---|---|---|
| = | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a pokud jsou si rovny, uloží zpět „1“, jinak „0“. |
| != <> | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a pokud si nejsou rovny, uloží zpět „1“, jinak „0“. |
| > | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a pokud je **B** větší než **A**, uloží zpět „1“, jinak „0“. |
| < | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a pokud je **B** menší než **A**, uloží zpět „1“, jinak „0“. |
| >= | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a pokud je **B** větší nebo rovno **A**, uloží zpět „1“, jinak „0“. |
| <= | 2 | 1 | Vyčte 2 hodnoty **A** a **B** ze zásobníku a pokud je **B** menší nebo rovno **A**, uloží zpět „1“, jinak „0“. |
| not | 2 | 1 | Vyčte hodnotu ze zásobníku a pokud je nenulová, uloží zpět „0“, jinak „1“. |
| and | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a pokud jsou obě nenulové, uloží zpět „1“, jinak „0“. |
| or | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a pokud je alespoň jedna nenulová, uloží zpět „1“, jinak „0“. |
| nor | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a pokud není alespoň jedna nenulová, uloží zpět „1“, jinak „0“. |
| xor | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a pokud je právě jedna nenulová, uloží zpět „1“, jinak „0“. |
| xnor | 2 | 1 | Vyčte 2 hodnoty ze zásobníku a pokud není právě jedna nenulová, uloží zpět „1“, jinak „0“. |

Tabulka 3: Logické instrukce

| Instrukce | Počet hodnot vstup | výstup | Popis |
|---|---|---|---|
| if | 2 | 0 | Vyčte label **A** a hodnotu **B** ze zásobníku a pokud je **B** nenulové zavolá místo odpovídající labelu **A**. |
| ifg<br>ifgo | 2 | 0 | Vyčte label **A** a hodnotu **B** ze zásobníku a pokud je **B** nenulové přejde na místo odpovídající labelu **A**. |
| ife<br>ifelse | 3 | 0 | Vyčte labely **A** a **B** a hodnotu **C** ze zásobníku a pokud je **C** nenulové zavolá místo odpovídající labelu **B**, jinak zavolá místo odpovídající labelu **A**. |
| ifeg<br>ifelsego | 3 | 0 | Vyčte labely **A** a **B** a hodnotu **C** ze zásobníku a pokud je **C** nenulové přejde na místo ve fimwaru labelu **B**, jinak na místo odpovídající labelu **A**. |
| jump | 1 | 0 | Vyčte label ze zásobníku přejde na místo odpovídající labelu. |
| call | 1 | 0 | Vyčte label ze zásobníku zavolá místo odpovídající labelu. |
| ret | 0 | 0 | Vykonávání firmwaru se vrátí na místo odkud bylo naposledy zavoláno. |
| set<br>store | 2 | 0 | Vyčte registr **A** a hodnotu **B** ze zásobníku a uloží hodnotu **B** do registru **A**. |
| drop | 1 | 0 | Vyčte hodnotu ze zásobníku a zahodí ji. |
| dup<br>duplicate | 1 | 2 | Vyčte hodnotu ze zásobníku a uloží ji zpět dvakrát. |
| swap | 2 | 2 | Vyčte hodnoty **A** a **B** ze zásobníku a zpět uloží nejdříve **A** a poté **B**, čímž tyto hodnoty v zásobníku přehodí. |
| sync | 0 | 0 | Instrukce spotřebuje veškeré interpretační kroky robota pro právě probíhající cyklus simulace. |
| nop | 0 | 0 | Prázdná instrukce spotřebovávající právě jeden interpretační krok. |

Tabulka 4: Řídící a ostatní instrukce

# 6 Překladač

Překladač (Compiler) je třída určená k překladu (Vyskočil, 2006) zdrojového kódu v RoboCodu umělé inteligence robota do vnitřního kódu (Firmware) robota, jež je srozumitelný pro interpretování chování robota interpreterem v průběhu klání. Třída Compiler používa třídu Tokenizer, jež slouží k lexikální analýze. Samotná překladač má pak na starosti syntaktickou analýzu a překlad do firmwaru.

## 6.1 Třída Tokenizer

Třída Tokenizer, jak už bylo řečeno výše, slouží k převodu zdrojového kódu v jazyce RoboCode na tokeny. Pro bližší představu Výpis 2 představuje ukázku kódu, jež se bude v průběhu překladu po jednotlivých analýzách měnit.

```
[Small Circle Robot]
                    # začátek
start:
  speed 3 + speed store
                    # otáčení o 45 stupňů
turn:
  0 reg00 store
  angle 45 + angle store
                    # kontrola radaru
wait:
  radar shooting ifg
  sync
  reg00 1 + reg00 store
  reg00 10 <= wait turn ifeg
                    # střelba
shooting:
  5 shoot store
  sync
  wait jump
```

Výpis 2: Příklad kódu v jazyce RoboCode

### 6.1.1 Průběh lexikální analýzy

Vstupem tokenizeru je text v jazyce RoboCode, který je podroben těmto krokům:

1. Odstranění komentářů z textu.
2. Vyčtení jména robota z hranatých uvozovek z prvního řádku. V případě, že robot nemá uvedeno jméno, je pojmenován „Unnamed“.
3. Parsováním vytvoření pole lexémů z textu.

Pokud v průběhu tohoto procesu došlo k vyvolání výjimky, je tato výjimka odchytnuta.

### 6.1.2 Výjimky

Výjimky definované ve třídě Tokenizer jsou typu TokenizerException. V případě, že je vyvolána některá z definovaných či nedefinovaných výjimek, je text této výjimky zobrazen uživateli v okně se zprávou. Definované výjimky:

**Jméno robota má špatný formát.** Tato výjimka je vyvolána v případě, že se někde na prvním řádku textu sice nachází znak „[“, avšak buď není na první pozici v tomto řádku a/nebo na poslední pozici není „]“.

**Jméno robota není vyplněno.** Tato výjimka je vyvolána v případě, že jsou sice prvním a posledním znakem na řádku „[“ resp. „]“, avšak místo mezi nimi neobsahuje text.

```
start : speed 3 + speed store turn: 0 reg00 store
angle 45 + angle store wait : radar shooting ifg sync
reg00 1 + reg00 store reg00 10 <= wait turn
ifeg shooting: 5 shoot store sync wait jump
```

Výpis 3: Seznam lexémů po lexikální analýze odřádkovaný po deseti

## 6.2 Třída Compiler

Třída Compiler provádí překlad z tokenizerem rozparsovaného RoboCodu do posloupnosti celočíselných kódů, jenž je po překladu vrácena do firmwaru robota.

### 6.2.1 Průběh syntaktické analýzy a překladu do firmwaru

Po lexikální analýze, o kterou se postaral tokenizer, přichází na řadu syntaktická analýza a překlad do firmwaru. Oba kroky probíhají zároveň v tomto sledu:

1. Pro každý token se provede:
   (a) Ověření, zda je token číslo. Pokud ano, a je v povoleném rozmezí, převést token na číslo a přejít na další token.
   (b) Ověření, zda je tokel label. Pokud ano, přidat do seznamu labelů, vyřešit odkazy na tento label a přejít na další token.
   (c) Ověření, zda je token registr. Pokud ano, převést token na kód registru a přejít na další token.
   (d) Ověření, zda je token odkaz na label. Pokud ano, dosadit pozici labelu a přejít na další token.
   (e) Ověření, zda je token operátor:
      - Pokud ano, je token nahrazen kódem odpovídajícím operaci.
      - Pokud ne, pak je nutno považovat token za doposud nevyřešený label a nahradit token kódem označujícím nevyřešený label.
2. Ověření, zda kód neobsahuje nevyřešený label.

Pokud v průběhu tohoto procesu došlo k vyvolání výjimky, je tato výjimka odchytnuta.

### 6.2.2 Výjimky

Výjimky definované ve třídě Compiler jsou typu CompilerException. V případě, že je vyvolána některá z definovaných či nedefinovaných výjimek, je text této výjimky zobrazen uživateli v okně se zprávou. Definované výjimky:

**Příliš krátký robocode.** Tato výjimka je vyvolána v případě, že délka textu po lexikální analýze je rovna nule.

**Byla překročena celková kapacita firmwaru.** Tato výjimka je vyvolána v případě, kdy byla dosažena maximální kapacita firmwaru a chceme přidat další kód do firmwaru.

**Číslo je mimo povolený rozsah.** Tato výjimka je vyvolána v případě, kdy je token rozpoznán jako číslo, avšak není v povoleném rozsahu.

**Label je příliš dlouhý.** Tato výjimka je vyvolána v případě, že název labelu je příliš dlouhý.

**Token není klíčové slovo, label, ani registr.** Tato výjimka je vyvolána v poslední části překladu, kdy se ověřuje, zda se ve firmwaru nalézá nevyřešený label.

```
30501 3 30000 30501 30028 0 32000 30028 30500 45
30000 30500 30028 31004 28 30026 30035 32000 1 30000
32000 30028 32000 10 30020 13 5 30027 5 30502
30028 30035 13 30032
```

Výpis 4: Seznam kódů firmwaru po překladu odřádkovaný po deseti

# 7 Interpreter

Interpreter firmwaru slouží k interpretaci kódu přeloženého překladačem z kódu v jazyce RoboCode. Neinterpretuje tedy přímo RoboCode kód. Interpreter pracuje po krocích a to tak, že v každém kroku zpracuje jednu položku z kódu firmwaru.

## 7.1 Counter, Stack a CallStack

### 7.1.1 Counter

Counter je ukazatel na konkrétní místo ve firmwaru a slouží k řízeným odskokům v rámci firmwaru.

### 7.1.2 Stack

Stack je hlavním zásobníkem interpreteru, do kterého se při interpretaci firmwaru ukládají hodnoty. Stack je omezen na 1024 položek.

### 7.1.3 CallStack

CallStack je vedlejším zásobníkem interpreteru a slouží k uchovávání indexů ukazatele při volání podprogramu. CallStack je omezen na 512 položek.

## 7.2 Krok interpretace

Každý krok interpretace se sestává z následujícího postupu:

1. Ověření pozice ukazatele.
2. Vyčtení kódu.
3. Ověření, zda je kód číslo.
4. Ověření, zda je kód registr.
5. Ověření, zda je kód operace.

### 7.2.1 Ověření pozice ukazatele

V této části se ověří, zda ukazatel ukazuje stále na místo ve firmwaru. Pokud ne, je vyvolána InterpreterException výjimka „Skončila interpretace robota“.

### 7.2.2 Vyčtení kódu

Do odpovídající proměnné je vyčten kód z firmwaru, nacházející se na pozici, kam odkazuje ukazatel.

### 7.2.3 Ověření, zda je kód číslo

V tomto kroku se ověří, zda je kód číslo. Pokud ano, následně se provede uložení kódu do Stacku. Pokud byl však Stack plný, je vyvolána InterpreterException výjimka „Zásobník je plný“.

### 7.2.4 Ověření, zda je kód registr

Zde se ověří, zda kód odpovídá některému z registrů. Pokud ano, je následně provedeno uložení tohoto kódu do Stacku, opět s ověřením kapacity Stacku a případným vyvoláním výjimky.

### 7.2.5 Ověření, zda je kód operace

V případě, že kód odpovídá kódu některé z operací, provede se příslušná operace. Ukázka zdrojového kódu operace If (základní podmínka) je vidět na Výpisu 5.

```
private int opIf()
{
    short tmp1 = stack.Pop();
    short tmp2 = stack.Pop();
    tmp2 = readNumberOrLoadRegister("If", tmp2);
    if (tmp2 != 0)
    {
        if (isInFirmware(tmp1))
        {
            if (callStack.Count == CALLSTACK_SIZE)
            {
                throw new OperationException("If", "CallStack je plný.");
            }
            callStack.Push(counter);
            counter = tmp1;
        }
        else
        {
            throw new OperationException("If", "Skok odkazuje mimo firmware.");
        }
    }
    return 1;
}
```

Výpis 5: Zdrojový kód operace If

## 7.3 Výjimky

Interpreter má 2 typy definovaných výjimek: InterpreterException a OperationException.

### 7.3.1 InterpreterException

Tento typ výjimky je vyvolán, pokud nastane jedna z následujících situací:

- Skončila interpretace robota.
- Zásobník je plný.

### 7.3.2 OperationException

Výjimka OperationException je vyvolána, pokud nastane chyba v provádění některé z operací. Tyto chyby jsou:

- Výsledek je mimo rozsah.
- Tangens z úhlu 90 nebo 270 nelze vypočítat.
- Zásobník je plný.
- CallStack je plný.
- CallStack je prázdný.
- Skok odkazuje mimo firmware.
- Ukládání do read-only registru.
- Ukládání mimo registry.
- V registru není číslo.
- Číslo není v rozsahu čísel.

# 8 Závěr

Výsledkem této práce je plně funkční simulátor bitev robotů AI Clash, který používá vlastní specificky navržený jazyk RoboCode, aby interpretoval chování robotů v bitvě. Implementaci tohoto simulátoru jsem si zvolil hlavně proto, abych si osvojil pravidla a postupy při implementaci překladačů a interpreterů. Navíc se mi tato práce zdála zajímavá jak po implementační, tak po vizuální stránce.V průběhu implementace jsem přicházel na různá další vylepšení, jenž souvisela se simulací, a která původně nebyla plánována. Některá z nich byla již implementována, jiná budou dodělána v budoucnu. Při implementování simulátoru jsem si postupem času osvojil dovednosti související s nejnovějšími verzemi jazyka C# a technologii .NET, které mi jistě poslouží při programování dalších prací.

## 8.1 Plánovaná vylepšení

I když jsem si postupem času postfixovou notaci osvojil, pro některé uživatele by toto mohlo být překážkou a tak je mým cílem do budoucna pro umělou inteligence robota vytvořit ještě druhý jazyk, který by měl jiná pravidla a mohl by případně uživatelům ještě více vyjít vstříc při programování chování robota.

V průběhu ukončování práce mě také napadlo, že by kolize, či jiné stavy nastávající v simulaci mohly u robota vyvolávat vyvolání výjimky a tedy by se v tom případě zavolala jiná část kódu bez ohledu na právě prováděnou. Uživatel by pak mohl zvolit, zda chce odchytávat tyto výjimky, či je ignorovat a zjišťovat stavy v simulaci pomocí registrů.

Posledním zásadním plánem je změna započítávání bodů a ocenění robotů. Dosud je tento výpočet založen na mém odhadu a úsudku, které situace vyzdvihovat a bodovat je tedy více a které se naopak snažit potlačit. Tyto výpočty budou v budocnu upravovány na základě zpětné vazby uživatelů, po nasazení aplikace a po mnoha testováních.

Tomáš Worek

# 9 Reference

[1] METCALF, John, *Darwin: Survival of the Fittest among Programs* [online],
Poslední aktualizace 27.1.2011a 13:40,
Dostupné na WWW:<http://corewar.co.uk/darwin/>

[2] METCALF, John, *RobotWar: the Battlefield of the Future* [online],
Poslední aktualizace 27.1.2011b 13:46,
Dostupné na WWW:<http://corewar.co.uk/robotwar/>

[3] JONES, D. G.; DEWDNEY, A. K., *Core War Guidelines* [online],
Poslední aktualizace 5.2.2006 18:00,
Dostupné na WWW:<http://corewar.co.uk/cwg.txt>, 1984

[4] VYSKOČIL, Michal, *Seriál: Jazyky a překladače* [online],
<http://www.abclinuxu.cz/serialy/jazyky-a-prekladace>, 2006